# Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 01-023

## Personalized Profile Based Search Interface With Ranked and Clustered Display

Sachin Kumar, B. Uygar Oztekin, Levent Ertoz, Saurabh Singhal,
Euihong (sam) Han, and Vipin Kumar

June 01, 2001

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **01 JUN 2001** | | **-** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **Personalized Profile Based Search Interface With Ranked and Clustered Display** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Army Research Laboratory,2800 Powder Mill Road,Adelphi,MD,20783-1197** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**The original document contains color images.**

14. ABSTRACT
**see report**

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | |
| **unclassified** | **unclassified** | **unclassified** | | **20** | |

# Personalized Profile Based Search Interface With Ranked and Clustered Display

Sachin Kumar, Levent Ertöz, Saurabh Singhal, B. Uygar Oztekin, Eui-Hong Han and Vipin Kumar
University of Minnesota, Department of Computer Science /Army HPC Research Center
Minneapolis, MN 55455
{sachin, ertoz, singhal, oztekin, han, kumar}@cs.umn.edu

**Abstract**

*We have developed an experimental meta-search engine, which takes the snippets from traditional search engines and presents them to the user either in the form of clusters, indices or re-ranked list optionally based on the user's profile. The system also allows the user to give positive or negative feedback on the documents, clusters and indices. The architecture allows different algorithms for each of the features to be plugged-in easily, i.e. various clustering, indexing and relevance feedback algorithms, and profiling methods.*

## 1. INTRODUCTION

Due to the explosive growth of the World Wide Web, a huge amount of information is available online. For example "Google" [2] claims to have a repository of over one billion web pages. Search engines make it possible to easily access this information. To retrieve the desired information from these pages, search engines maintain an inverted index of these web pages, and in response to the user's query, all the documents that match the query terms are returned. Very often, a large number of web pages match query terms, but only few of those may be of interest to the user. This problem is particularly bad when the user query contains only a few terms and thus may refer to a broad topic. Search engines do return web pages in some rank order determined by their match to the query, and in some case by the importance of the web pages. Nevertheless it is often impossible for a search engine to know which web pages may be most relevant to the user. Since a user typically looks only at the first few items returned (that are available on the first or second screen returned by the search engine), results further down in the ranking are never looked at, even if they contain the desired information.

There are many solutions to this problem. One possible approach is to customize the ranking of returned web pages for each user so that the top ranked documents more closely resemble the taste and interest of the user. A profile can be built for each user based upon the past queries and corresponding web pages visited. This profile can be used to influence the ranking of the pages returned in upcoming queries. For example, for search query on "language", such a system may highly rank web page referring to computer language for a user with computer science/information technology background. In contrast, for a user with liberal arts background, it may provide higher ranks to web pages referring to natural languages. Thus when profile is used to influence the returned documents, the search results are personalized and are dynamic with respect to user. The idea of profile-based ranking can be extended to groups of users with common interest or those working in an organization. The concept of user profiles (based upon the past queries) has been used in various collaborative search engines [10,11,12,14] and in other interesting web applications [6,13,20].

Document clustering is another promising method for organizing search results. Instead of displaying the documents in a ranked order, we might present to the user different themes emerging from the documents in the form of clusters of web pages. Now the user can find the desired information by choosing the cluster of his interest. Clustering based approach to documents browsing has been investigated extensively [4, 5, 9, 16], and has been incorporated in several meta-search engines, e.g. Grouper [17, 18, 19], MSEEC [8], Manjara [22].

One major drawback of the clustering based algorithms is that a cluster label often fails to provide a complete picture of the web pages contained in the cluster. If the label contains only a few words, then it may show no information about some documents contained in the cluster. If the label contains too many words, it may fail to identify the underlying theme. Since each web page is part of only one cluster, the user may miss the web page of interest if its contents are not reflected in the cluster label.

An alternate method is to present web pages organized under indices. If the index is in the form of hierarchy, it is easy to navigate through a large result set. By clicking on a keyword from the index, user can find all documents that contain that keyword. This method is used in meta-search engines Vivisimo [http://www.vivisimo.com], and Infogistics [http://www.infogistics.com].

When user does not find exactly what he is looking for in the search results, relevance feedback is one of the useful features. Using this, he can give feedback to the system about his likes and dislikes. Based on this feedback, a new query is generated automatically and new set of documents is presented to the user, which are likely to be of his interest. Some type of relevance feedback is implemented in variety of search engines like Manjara[22] and Google [2].
In our research group, we are developing Scout, a search engine interface that incorporates the above methods for presenting retrieval results along with relevance feedback mechanism. This system allows experimentation with different ways of incorporating profile information, clustering methods, index generation methods, and relevance feedback mechanisms and thus serves as a test bench for evaluating these concepts and algorithms. In this paper we describe the architecture of the system and demonstrate its performance in the context of web search. The system is available online at http://myscout.cs.umn.edu

## 2. Scout Architecture

Figure-1 shows the basic architecture of Scout meta search interface. Different options are accessible through various control buttons and popup menus. The user is allowed to modify the variables that influence the underlying algorithms.

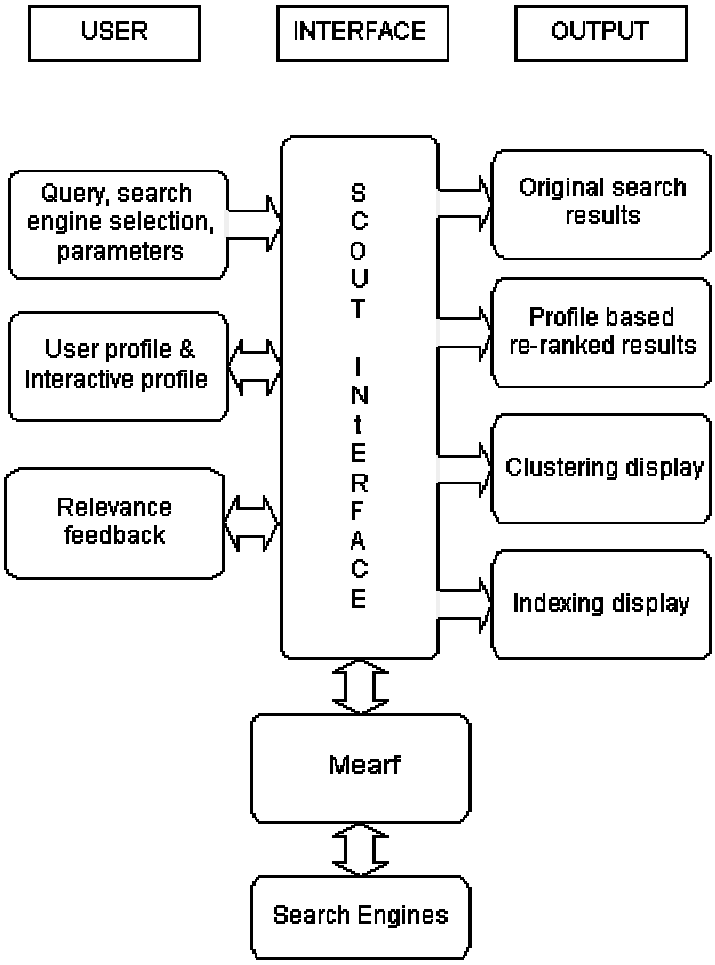


Figure 1. Architecture of Scout search engine

Through Scout interface, user's query is directed to mearf which fetches the results from one of these primary search engine based on user's options: Altavista, Directhit, Google, Excite, and Yahoo. Scout retrieves output of the search engine as a list of ranked snippets. Scout displays the results in one of the four different ways selected by the user. First is original documents fetched from the mearf without any processing. Second way of presenting results is profile based reranking. Clustering is another way in which results can be displayed, and final option is indexing.
In all kind of presentations user is provided with a checkbox to give positive or negative relevance feedback to the system, which is used to reformulate the query and fetch the new result set from the search engine.

In Scout, profile based ranking, clustering, and index generation are all done only using snippets. Every new user who visits the system is assigned a userid. This unique number is stored at user's system as a cookie. For each user, the list of queries done, and snippets of each corresponding URL visited are stored at Scout server as user's profile. If the same user visits the Scout system again, its userid is retrieved from his/her system, and the user's profile is pulled from Scout server. Scout also allows group profiles in addition to user's individual profile. In the current

implementation, all users belong to one global group but in future we plan to extend it to multiple groups, providing the user with an option to select his own group based on his interests.

We use vector space model to represent snippets [15], modify weights of the terms by using the TFIDF formula.

### 2.1 Mearf

Scout is powered by a special version of Mearf, our meta-search engine (http://mearf.cs.umn.edu) to handle all interactions with supported search engines. The module is capable of connecting to search engines, retrieving as many links as demanded by opening as many connections as necessary to the search engines included in the query in a transparent manner. It has a flexible and easy to update server configuration interface through which we can update the configuration data for a search engine if its html structure has changed, add new engines, or remove existing ones with minimal effort. It has an intelligent advertisement removal and duplicate elimination module embedded identifying majority of server specific advertisement or sponsored links and merging a grand majority of duplicate links with very little false positives. When asked to retrieve n links in total, it calculates approximately how many links should be retrieved for each search engines by considering number of search engines selected and guessing approximately how many links in total will be duplicate links. In most of the cases, it does not retrieve less than or more than 10% of the required number of links.

With this framework Scout is able to ask Mearf hundreds to a few thousands of links depending on the number of search engines included in the query. Note that maximum number of links that a typical search engine is willing to supply for a given query is about 300 to 800 links. Mearf's html retrieving module is fully multithreaded, it retrieves all the htmls, parses them, extracts links, processes them to convert to a format Scout and further algorithms (clustering and indexing algorithms for example) can easily process. Although Mearf could be used to retrieve links from a single search engine, its true power is in the methods it uses to re-rank and merge results coming from different search engines. Past logs and user click statistics suggested that Mearf scores and rankings are quite reliable [1] and even comparable to some of the best search engines available on the Web, supplying Scout a good deal of high quality links to be used further in subsequent modules in the pipeline.

### 2.2 Profile based reranking

Profile is a vector of weighted terms. To get this vector, we take the present query and match it against old queries stored in the user's profile. If there is a partial or full match between them, all the snippets who's URL were visited under those past queries are taken from the profile. Of these top k most related snippets to the query (judged by cosine measure) are chosen to take part in profile creation. Centroid of these k snippets serves as the profile vector for the user. Similarly a profile vector is created for the group that the user belongs to. Now the documents returned by the selected search engine can be reranked based on cosine measure of these profile vectors and original documents. The final rank of documents as shown by Scout is a hybrid of the original ranking, ranking due to the similarity with the user's own profile vector, and ranking due to

similarity with the group's profile vector. Interactive profile can also be used to rank the documents in the same way as users profile.

### 2.3 Clustering

Scout interface allows us to plug-in and experiment with any clustering algorithm with ease. The interface allows the user specify the different parameters of the clustering algorithm chosen, which gives us the opportunity to fine-tune the algorithms implemented.

The input to the clustering algorithm is the list snippets returned by Mearf along with their score. For each snippet, the sequence of word ids, the start and end locations of the word in the character sequence of the snippet and the stemmed form of each word is supplied to the clustering algorithm. The query itself is also passed, in case any clustering algorithm makes use of it. If the clustering algorithm has parameters that can be changed by the user, Scout passes these to the algorithm too. The output of the clustering algorithm is expected to be in the form of a graph. The graph is passed to the interface by specifying the name of the cluster along with its list of snippet numbers, and its parent and child information. In the case of a flat list, all the clusters will have the root node as its parent and they won't have any child nodes.

For a clustering algorithm to be useful in web search context, not only the clusters have to contain document that form a theme, but also they have to have a good label that gives a good description of the cluster without being too specific. Moreover, when a partitional algorithm is used for clustering, there is only one path from the top level to the related snippets the user is looking for, whereas if a clustering algorithm that allows overlaps between clusters is used, there can potentially be more than one way to reach the desired snippets.

Currently k-means, bisective k-means and shared nearest neighbor (SNN) clustering algorithms are implemented in scout. K-means is a well-known algorithm and won't be discussed here. Bisective k-means is a variation of k-means where one cluster is split into two in every iteration. The split for a cluster is determined by trying several k-means clusters with k=2 and choosing the best split among those. The cluster to be split can be chosen as the largest cluster, the least coherent cluster or by any other reasonable criteria. SNN algorithm uses the overlap between the nearest neighbors of two snippets as a measure of their similarity. It has a noise removal mechanism built-in and allows overlapping clusters, but the amount of overlap is usually very small.

K-means and bisective k-means have linear complexity. Although the SNN have a complexity of $O(n^2)$, the linear term dominates when the number of snippets the user requests is smaller than 1000, which is almost always the case, and it doesn't take more than 1 second to process 1000 snippets.

For k-means and bisective k-means, the summaries for the clusters are generated by picking up the most dominant several terms from the centroid. For SNN algorithm, the summaries are generated by using sequential phrase generation using the words that appear frequently in the cluster.

Vivisimo [3] uses a form of conceptual clustering [7] where they label the clusters as soon as they are formed and if a cluster does not allow for a good description, the cluster is rejected. Therefore the labels of the resulting clusters are, in general, good.

Most commonly used clustering algorithms in text clustering are k-means / k-mediods, hierarchical clustering and SVD based methods. All of these algorithms are partitional, however there are some variations which allow partial belongings to a cluster, i.e. fuzzy clustering.

The problem with partitional algorithms is that, every snippet returned by the search engine has to belong to a cluster. Search results contain noise and the amount of noise increases as more snippets are retrieved. For a clustering to perform well in snippet domain, it has to be able to deal with noise, probably by detecting and discarding the noise snippets.

Hierarchical clustering algorithms, in general, result in binary trees since two clusters are merged at a time (or one cluster is split into two). The resulting dendograms are hard to navigate due to the binary splits. Both hierarchical and SVD based algorithms have a run-time complexity $O(n^2)$ which used to make it unfeasible for online clustering. Considering that a user will not request thousands of snippets, and the size of the snippets are small, these algorithms can be used online.

### 2.4 Indexing

We can experiment with different indexing algorithms by plugging them into scout just as in clustering. The input and the output format are same as in clustering.

In index-based representation, when the user clicks on an index term, all the snippets containing that index term are presented to the user. A good indexing scheme should represent most of the snippets, if not all, without repeating itself too much. Some overlap between the snippets covered by the indices is useful since it provides the user several paths to desired snippets.

Currently, frequent phrase generation method is implemented in scout with a couple of variations. The first one is brute force sequential phrase generation, only using the words that have a support of 2 or more. The second one makes use of word clustering and uses the words that are in the same cluster to generate the sequential phrases. Once the phrases are generated, redundancy check is done, i.e. sub-phrases of a phrase with similar support are deleted, and if the support of the sub-phrase is considerably larger, then a branch in the hierarchy is created for the longer phrase.

Using indexing, it is easy to generate a reasonable hierarchy. The most important phrases with sufficient coverage can be placed at the top level and the indexing can be applied recursively until the number of snippets in a branch is relatively small. The second method described above is implemented recursively too.

Hierarchical representation of phrases is easier to navigate compared to a flat list. More information can be put into a hierarchy without confusing the user since the user will not see anything more than what he wants to see.

Applying natural language processing to the snippets is another way of obtaining phrases from search results. One can choose to display only the noun phrases to the users since they are better descriptors. Yet another way of doing indexing is to make use of the titles of the web pages. Since the titles are human generated according to the content of the page, they are intuitive and probably as good as it gets. However, there's no guarantee that there will be sufficient overlap between the titles of urls for a given query to be able to represent the results in a compact way.

### 2.5 Relevance Feedback

Scout system allows users to give their likes and dislikes with help of 3 way checkboxes in front of every snippet and cluster / index labels. On hit of a Go button, using this feedback a new query is formed, and is sent to the system for new set of results. At present Rocchio's Algorithm [21] is used for creating the new query based on feedback, which is given here in brief.

$Q_1 = Q_0 + \beta \Sigma R_i/n_1 - \gamma \Sigma S_i/n_2$, where
$Q_1$ = the vector for the final query
$Q_0$ = the vector for the initial query
$R_i$ = the vector for the relevant document i
$S_i$ = the vector for the non- relevant document I
$n_1$ = the number of relevant documents chosen
$n_2$ = the number of non-relevant documents chosen
$\beta,\gamma$ = weights to adjust the weights of relevant and non-relevant document vectors.

In future, we plan to play with more such relevance feedback algorithms. User can also see the reformulated query before sending it to the system, and also has the capability to modify the query, and add new terms to it.

## 3. Experimental evaluation:

To evaluate the features of Scout we tried a variety of queries in it and found that for certain type of queries, the features provided a better and easier way of finding the desired result. To start with, the user's past profile as well as interactive profile can be used to influence the ranking of results. For example on giving the query 'language', the top few documents in original result set had nothing to do with computer or programming languages. But if the user is of computer science background and has given programming related queries in past, his profile will have the related terms which will steer the results towards documents having to do with programming languages. Also, the user can interactively specify words to influence the rankings. As shown in figure 2, after selecting words 'xml', 'markup', 'java' from the profile and specifying 'perl' as a new word to influence ranking, the top documents in reranked result consisted of xml and java related topics though their original ranking was as high as 54 and 88.

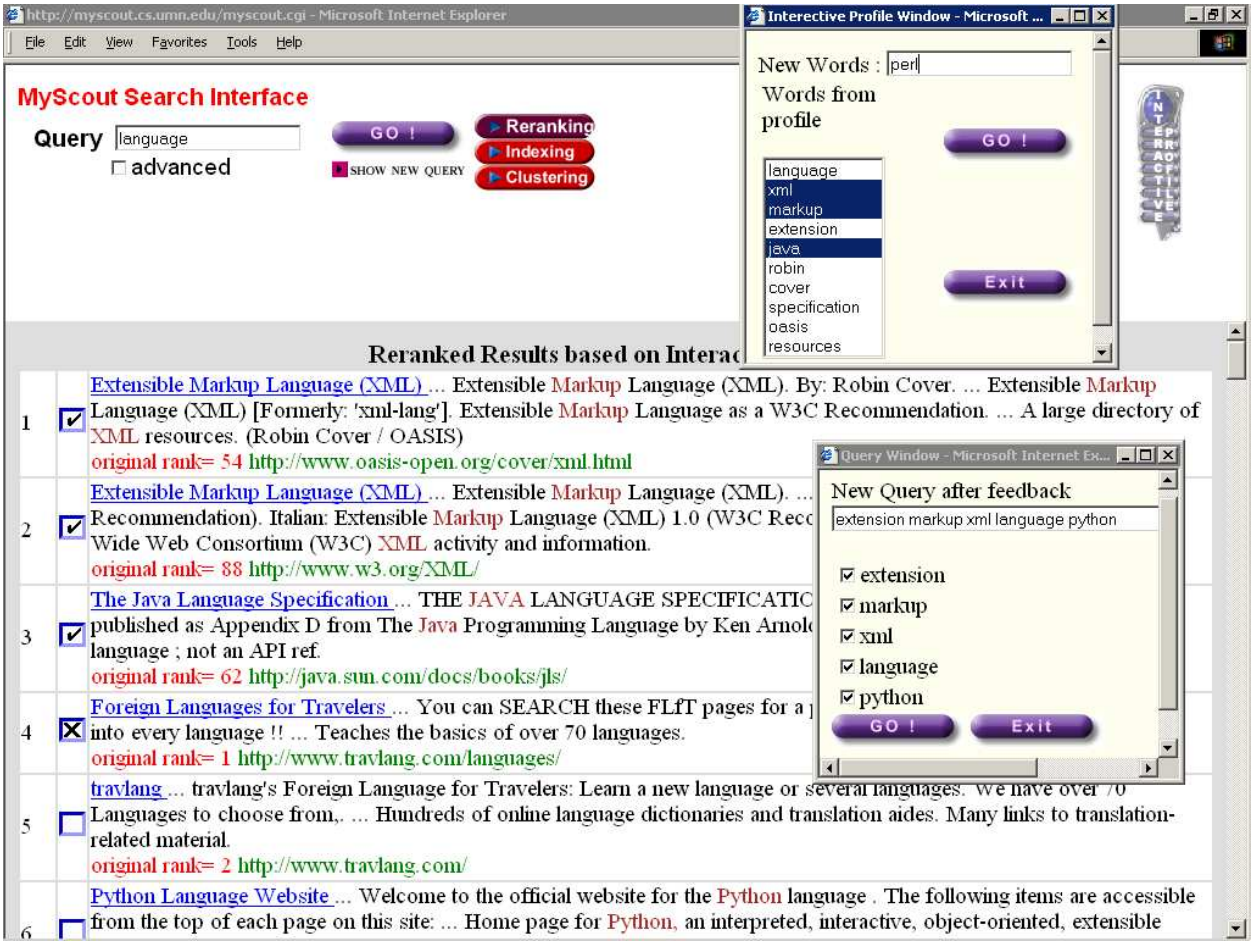


*Figure 2*: Showing influence of user past profile and interactive profile on the ranking of results. All the results have an option of providing relevance feedback and reformulating the query.

We can also use relevance feedback to reformulate the query. As shown in figure 2 we selected XML, and java related documents as positive feedback, and foreign language related document as negative feedback. On pressing the "Show New Query" button, the reformulated query is shown in a new window with the option of editing the newly added words, which on submission fetches a new result set having all the programming related hits as shown in figure 3.

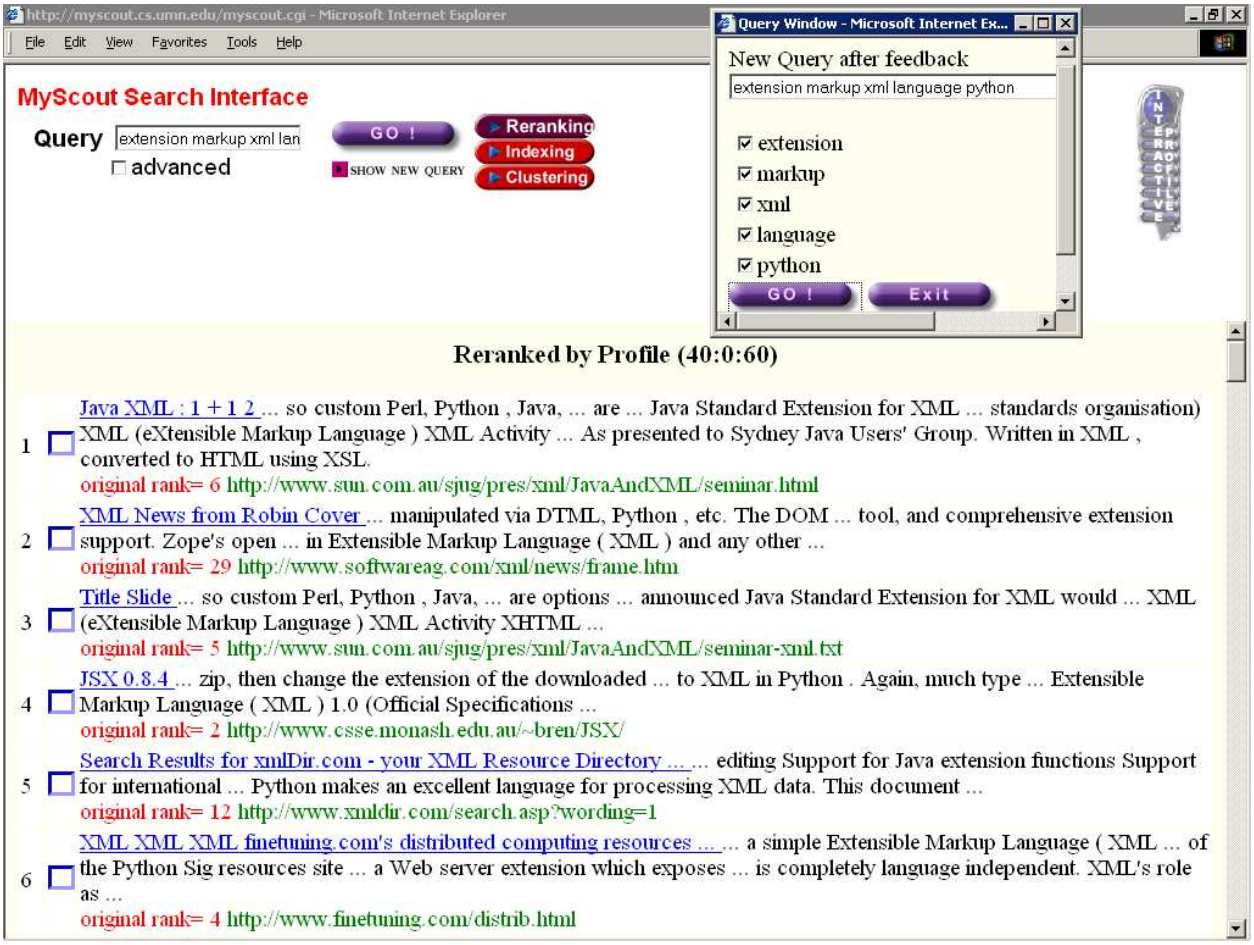


*Figure 3*: Showing the use of relevance feedback where the user reformulates the query and gets new result set with only desired type of documents.

For generic queries, indexing and clustering proved to be quite effective. For the query "computer science", the original results returned by other search engines had a jumble of various diverse topics like computer science departments, computer science reports, computer science research work etc. But after doing indexing on this result set, all these different topics were separated out and were listed under singular indices as shown in figure 4. Using these indices it's a lot easier to focus on the desired topic and see the relevant results simply by clicking on the corresponding index or sub-index. The only snag was that some insignificant words like 'contact', 'fax' etc. were so common among documents that they ended up as indices in themselves. This is really not a problem because as long as the number of irrelevant indices is not overwhelming they will not be distracting for the user you can simply chose to expand the relevant indices. Infact he can even use relevance feedback option to select the desired topic or deselect an undesired one and thus modify the query to get better results on the chosen topic.

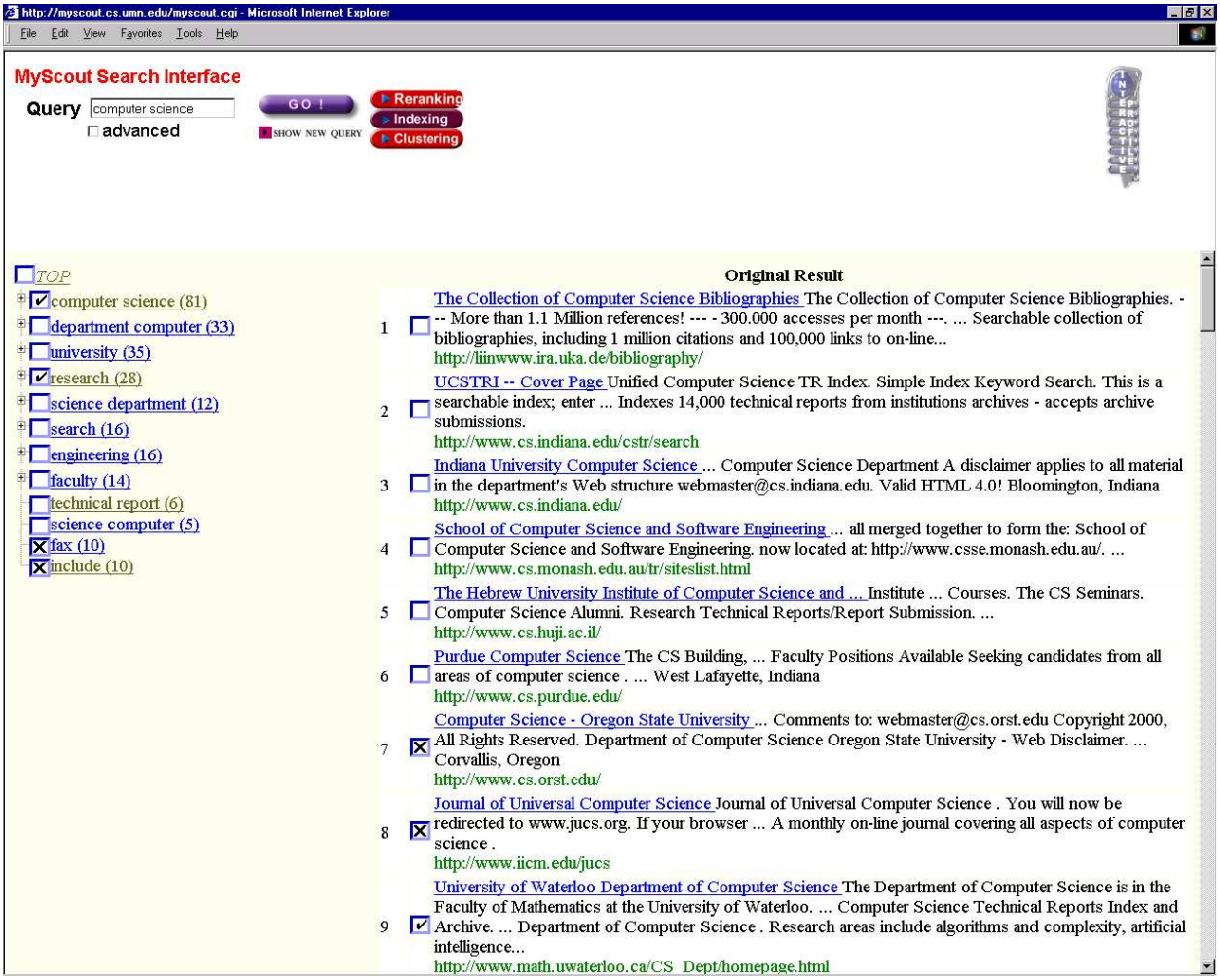


*Figure 4*: Showing indexing on query "computer science" along with original results for this query. The indices shown are arranged as a hierarchy, which can be expanded or collapsed and can also be used for relevance feedback.

Clustering was equally powerful for generic queries and separated the variegated topics into separate clusters as shown in figure 5. The added advantage of clustering over indexing was that there were no clusters on inconsequential words and the cluster labels were more descriptive to the projected topic than they were in indexing. But the downside here was that number of documents under each topic (represented as cluster) was not large as in indexing, as all clusters are disjoint for 'partitional' clustering algorithms and thus will not have any common or overlapping documents.

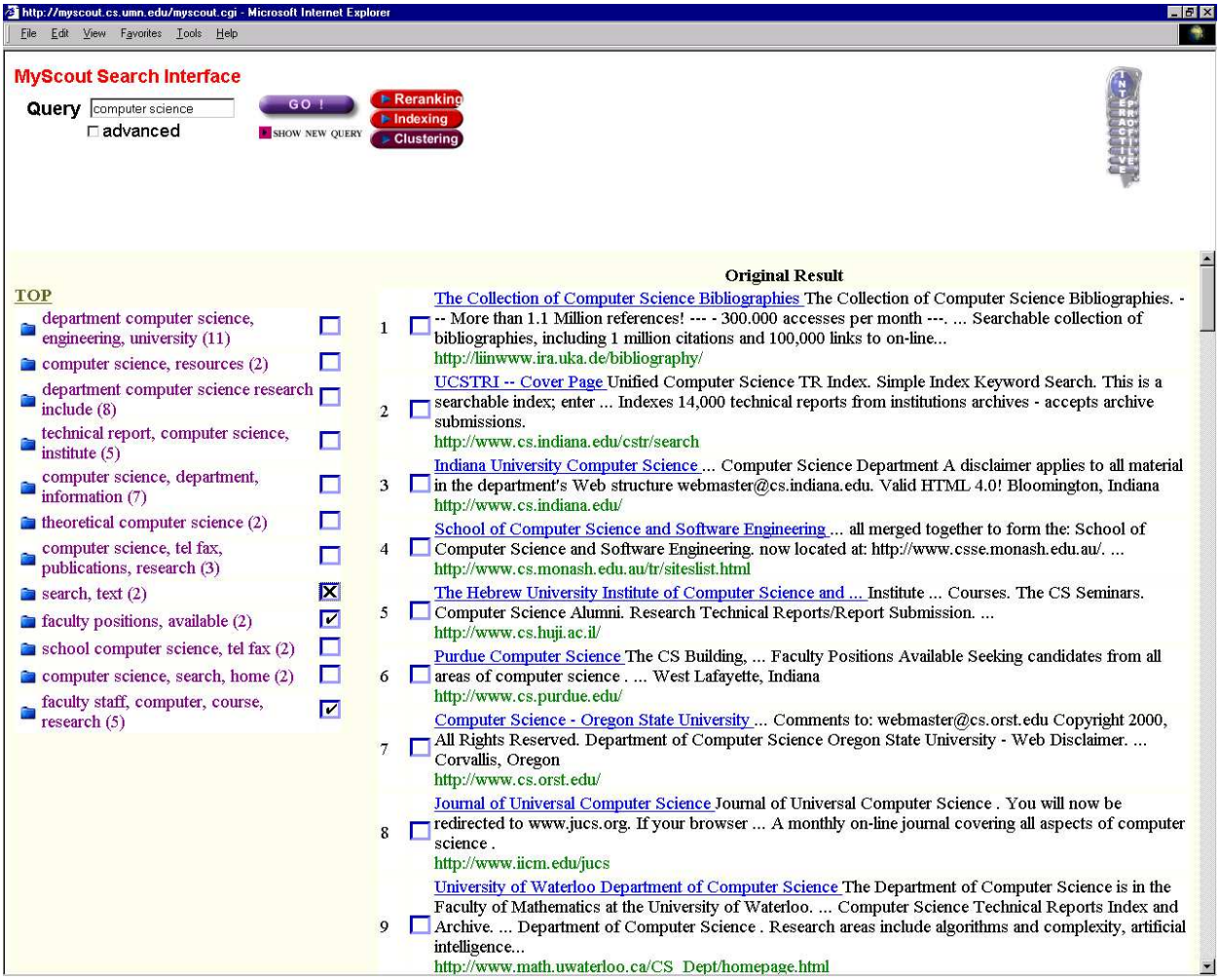


*Figure 5*: Showing clustering on query "computer science" along with original results for this query. The documents in each cluster can be obtained by clicking on the folder icon of relevant cluster.

Therefore indexing and clustering are really good features for using on generic queries where user is not sure of exact document he/she is looking for, but if user has something specific in his/her mind and gives rather long query in its pursuit then things are a bit different. Here, if the original results has the desired document among top ten results then using indexing/clustering wouldn't help much but if that sought after document is buried somewhere deep in the result set then using hierarchical index scheme its much easier for user to navigate through the appropriate indices and reach for the document.

Indexing and clustering are also very handy for concept-based search queries. In this situation a set of words can denote multiple concepts. To know the concept (or the context) in which the words appear, the neighboring words need to be examined and the presence of a certain type of nearby words signifies a certain concept denoted by document as a whole. For example for the query "association mining", the top 40 hits in original result set were related to metallurgy mining and related stuff and had no mention of association rules as in data mining scenario. But after using indexing on this result set, there was not only a separate index on 'rules→association

rule mining', but in fact the metallurgy mining was better organized with indices on 'mining industry', 'mining associations' etc. (figure 6) The same was true for clustering with a separate cluster appearing on 'association rule mining' (figure 7).

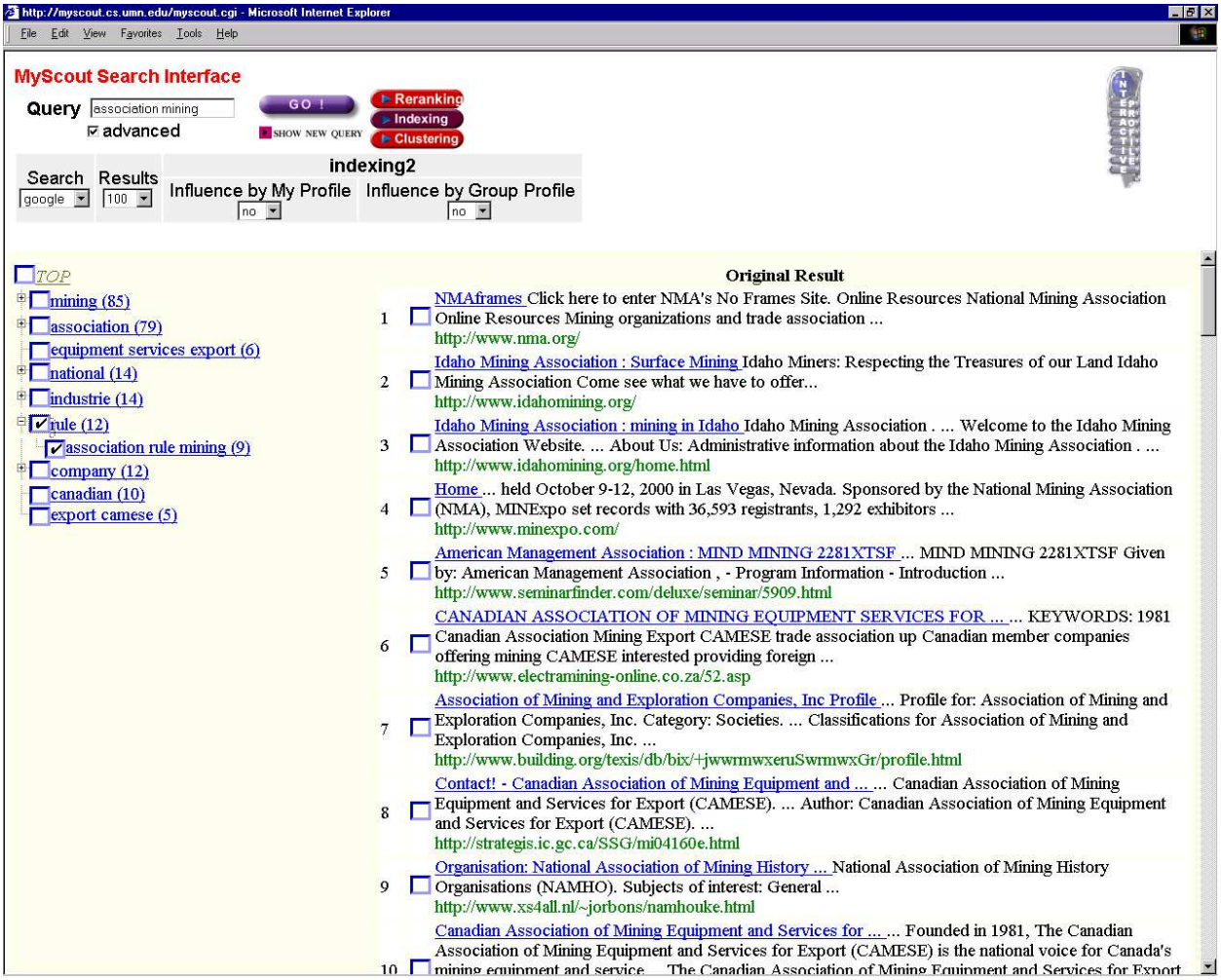


*Figure 6*: Showing indexing on query "association mining" along with original results reported for this query (note that top10 hits do not have anything on association rule mining)

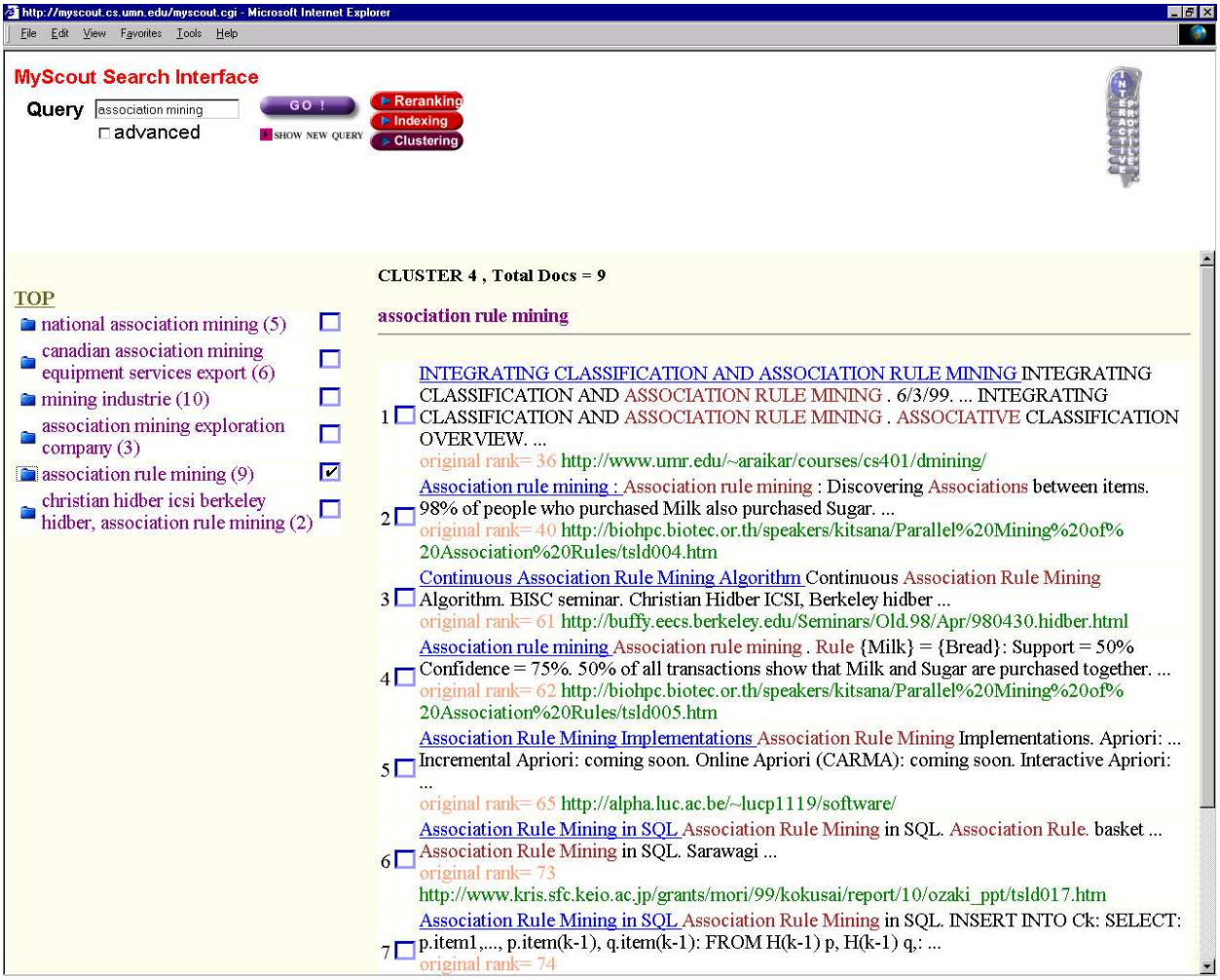


*Figure 7*: Showing clustering on query "association mining" along with documents contained in 'association rule mining' cluster.

Another example for concept based query given was 'raging bull' for which original result had potpourri of results from stock market, movies etc. Doing indexing and clustering on results organized them into distinct indices/clusters of possible concepts allowing the user to select the concept of his/her choice and dwell on it further. By not restricting to query keywords and observing neighboring words, indexing and clustering are immensely successful in bringing out all the related notions with the query. For a query like "search engines" the result set is neatly structured in to indices/clusters of 'search directories', 'meta search engines', 'search guides' and 'list of search engines' etc. Similarly for query "sports", the index/cluster labels consisted of various types of sports like basketball, football, golf and hockey etc. thereby making it really easy for users to navigate through the results.

The efficacy of indexing and clustering is also seen in polysemic queries where a same word has multiple meanings depending upon the context it's used in. Like for the query "intelligence", the top few documents in the original result were mostly on 'artificial intelligence' and documents with other meanings were obscured down the result set. After clustering, there were separate

clusters on ‘business-’, ‘artificial-’, ‘military-’ intelligence etc. as shown in figure 8. The results with indexing were similar though labels were not as intuitive as in clustering.

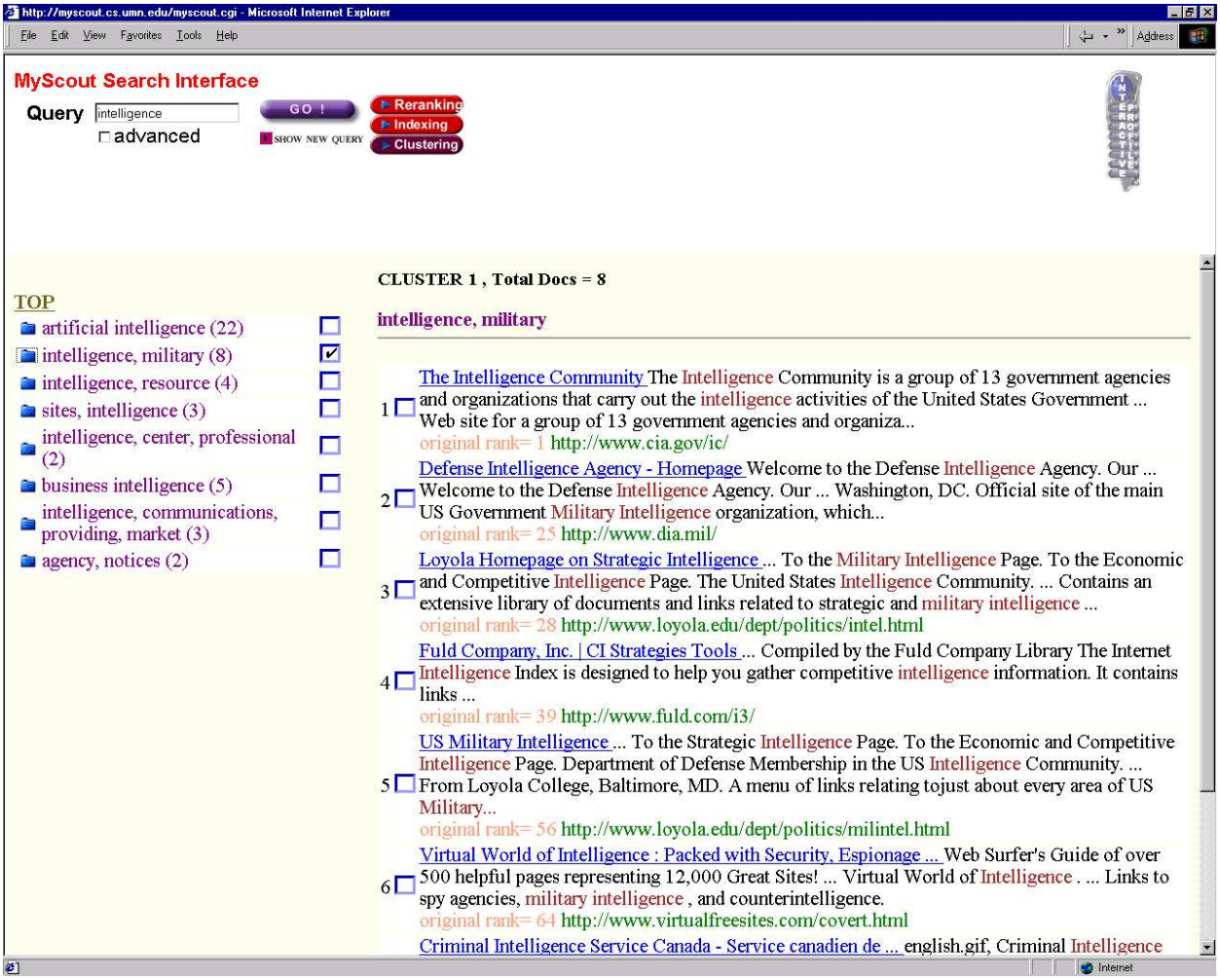


*Figure 8*: Showing clustering on query “intelligence” along with documents contained in ‘intelligence military’ cluster.

Indexing and clustering also help in the problem of synonymy (different words having same meaning), by identifying the synonyms to given query and making unique indices or clusters of them. For example for the query “cars” the algorithms were able to pick ‘auto’, ‘vehicle’ etc. resulting in more comprehensive presentation of results to the user.

Another advantage of using indexing is that it facilitates natural language queries by making phrase searching very easy and efficient. For the query “knowledge discovery process”, the original result set from search engines contains documents in which either one or combination of query words is present and that too on any order (or sequence). To trace the documents that have complete phrase in them from this result set is an uphill task. Using indexing we have a separate sub index having ‘knowledge discovery process’ as the label, clicking on which we can retrieve

all the documents that deal with the complete phrase (in same sequence of query words), instead of dealing with just one or more query words. (figure 9)

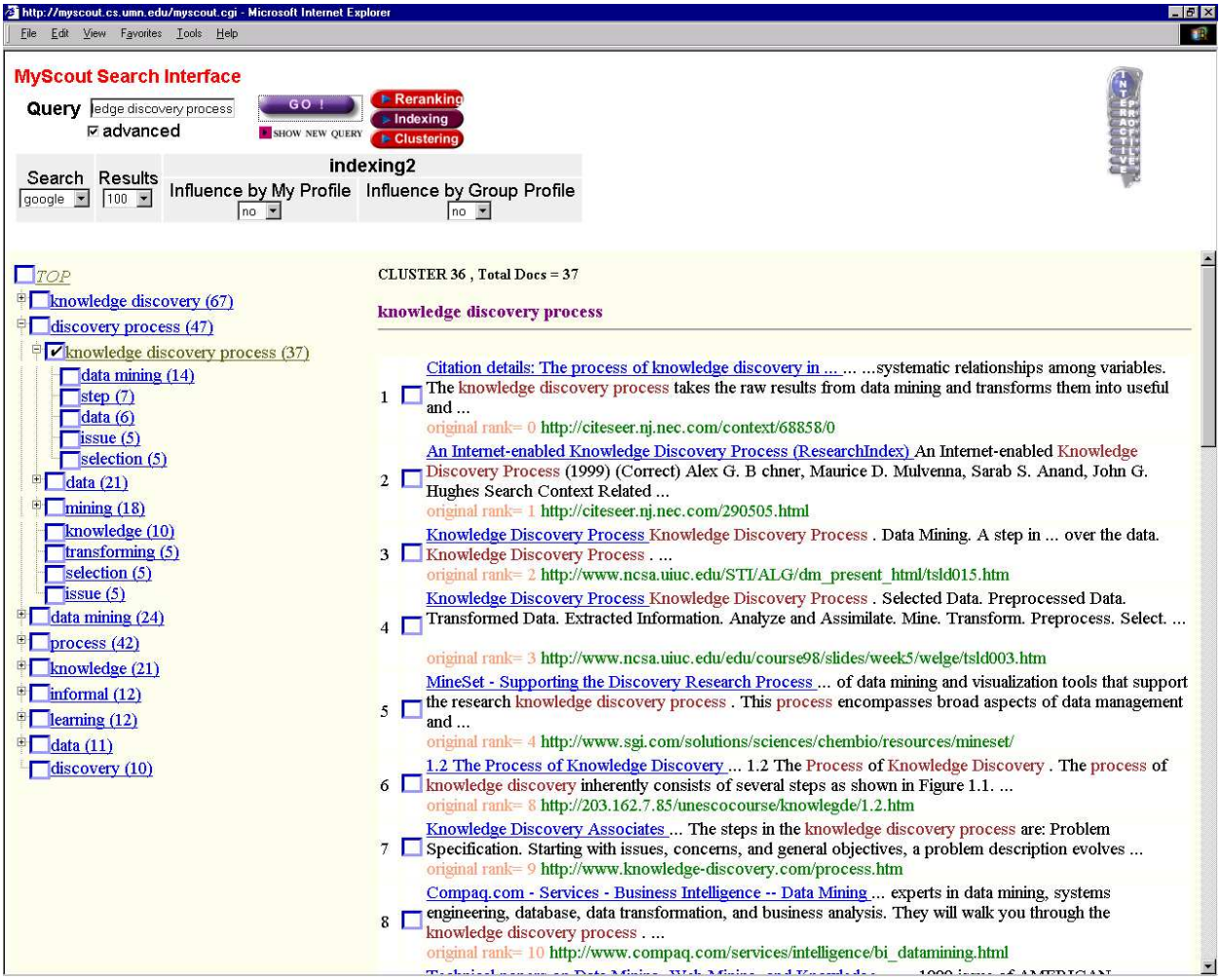


*Figure 9*: Showing indexing on query "knowledge discovery process" along with documents contained in the corresponding sub-index.

## 4. Conclusion and future work

We have presented the architecture of Scout. This system allows the user to more effectively explore a large number of search results returned by search engines and also allows the presentation of results to be influenced by the profile of the user and its group (as represented by the past queries).

The ultimate goal behind this project is to make it easier for the user to retrieve information, starting from searching to display, and display to relevance feedback. At this time, the system uses only snippets for generating clusters profiles and dynamic indices. Since snippets do not provide an exact picture of the document, resulting rankings, clusters, and dynamic indices are

not very accurate, as they could be, if the complete documents were used. Snippets have been used in Scout to make it fast. Another possible solution is to add a complete in-house search engine to Scout.

A number of items need further work. Current scheme for maintaining user profile and group profiles are quite simple, and new scheme needs to be investigated. The quality of cluster labels is highly dependent upon the ability of the phrase extraction module to detect quality phrases.

Profile information can also be used to influence the generation of dynamic indices. One possibility is to create an orthogonal index using the documents in the profile, and some of these to augment the index created using the snippets returned in response to the query.

## 5. Acknowledgements

This work was partially supported by the Army High Performance Computing Research Center, contract number DAAH04-95-C-0008. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by AHPCRC and the Minnesota Supercomputing Institute.

## 6. References

[1] Oztekin B.Uygar, George Karypis, Vipin Kumar. "***Expert Agreement and Content Based Reranking in a Meta Search Environment using Mearf***", TR Department of Computer science, University Of Minnesota.

[2] Brin S., Page L., ***The anatomy of a large-scale Hypertextual web search engine,*** Stanford Univeristy, 1998.

[3] Valdes-Perez, R. E., Pesenti, J. and Palmer, C. R. "***Hierarchical Document Clustering for Digital Libraries***".

[4] Cutting Douglass R., Karger David R., Pedersen Jan O. and Tukey John W., ***Scatter/Gather: a cluster-based approach to browsing large document collections***. Pages 318 – 329.

[5] Cutting Douglass R., Karger David R., Pedersen Jan O. ***Constant interaction-time scatter/gather browsing of very large document collections.*** Pages 126 – 134, 1993.

[6] Evangelos P. Markatos, Tziviskou C., Papathanasiou A., ***Effective Resource Discovery on the World Wide Web*** Institute of Computer Science (ICS), Foundation for Research & Technology -- Hellas (FORTH), Crete, Copyright 1998.

[7] Michalski, R.S., and Stepp, R. E. "***Learning from observation: Conceptual clustering***", Machine Learning: An Artificial intelligence approach, pp. 331-363, Morgan Kauffmann, San Mateo, CA, 1983.

[8] Hannappel P., Klapsing R., Neumann G., ***MSEEC - A Multi Search Engine with Multiple Clustering*** Information Systems and Software Techniques, University of Essen, Universitätsstraße 9, D-45141 Essen, Germany.

[9] Hearst Marti A., Pedersen and Jan O., ***Reexamining the cluster hypothesis: scatter/gather on retrieval results.*** Pages 76 – 84, 1996.

[10] Lieberman H., Dyke Neil V., Vivacqua A. ***Let's Browse: A Collaborative Web Browsing Agent*** MIT Media Lab, Cambridge, MA, USA, {lieber, nwv, adriana}@media.mit.edu

[11] Mladenic D., Harper David J., Mechkour M., ***Text-learning and related intelligent agents (1999)*** .

[12] Moukas, A. and P. Maes, ***Amalthaea: an evolving multi-agent information filtering and discovery systems for the WWW***." Autonomous agents and multi-agent systems. Vol. 1, pp. 59-88, 1998.

[13] Pitkow James E., and Krishna A. Bharat***. WebViz: A tool for world-wide web access log analysis.*** Proceedings of the First International WWW Conference, 1994.

[14] Rucker J., Polanco Marcos J., ***SiteSeer, Personalized Navigation for the web***, Communications of the ACM, pp. 73-75, Vol, 40, no. 3, March 1997.

[15] Gerard Salton, A. Wong, & C.S. Yang***, A vector space model for automatic indexing***. *CACM* 18, 1975, 613-620.

[16] Silverstein C., Pedersen Jan O., ***Almost-Constant-Time Clustering of Arbitrary Corpus Subsets.*** SIGIR 1997: 60-66, [DBLP:conf/sigir/SilversteinP97] 1997 .

[17] Zamir O., Etzioni O, ***Grouper: A Dynamic Clustering Interface to Web Search Results.*** WWW8., 1998.

[18] Zamir O., Etzioni O, ***Web Document Clustering: A Feasibility Demonstration*** SIGIR 1998: 46-54 [DBLP: conf/sigir/ZamirE98], 1998.

[19] Zamir O., ***Fast and Intuitive Clustering of Web Documents***. Qual's Paper, University of Washington. 1997.

[20] Caesar S., Chugh M., Aggrawal P., **Transparent *Development of User Profiles with relevance to Web-Portal Design*** Department of Electronics and communications Engineering, Netaji Subhas Institute of Technology, formerly Delhi Institute of Technology University of Delhi.

[21] Rocchio J. J., "***Relevance feedback in information retrieval*** ". In The SMART Retrieval System-- Experiments in Automatic Document Processing, pp. 313-323, Englewood Cliffs, NJ, 1971. Prentice Hall, Inc.

[22] Manjara: ***Fast Comprehensive Keyword Summary and Classification*** http://cluster.cs.yale.edu/about.html